

Mars Exploration Rover Vision Data Analysis: Dust Concentration of Mars Dust Devils

Rebecca Arvanites*

NASA Glenn Research Center, Cleveland, OH, 44135

Understanding the properties of dust devils is important for gaining knowledge of the Martian environment. The goal of this project is to investigate the distribution of dust concentration in dust devils on Mars, using the Navcam images taken by Spirit. The dust concentration of the dust devils is examined by determining the opacity, τ , which is a non-dimensionalized parameter describing the amount of dissipation that the dust devil causes to light traveling through it. The distribution of τ was investigated to determine how τ varies with position within the dust devils, and how τ changes with time as the dust devils develop. One challenge was distinguishing the dust devils in the images from the background for the portion of the dust devil in front of the sky. Also of interest is how the optical properties of the dust in the devil compare to those of the dust suspended in the atmosphere; calculations were done first assuming the same optical properties and then more rigorously without these assumptions.

Nomenclature

τ	= dust opacity
I	= observed light intensity
I_0	= initial light intensity, before dissipation through atmosphere or dust
I_d	= light intensity observed by line of sight through dust devil to the ground
I_{sd}	= light intensity observed by line of sight through dust devil to the sky
I_{sky}	= light intensity of the sky

I. Introduction

DUST is an important part of the Martian environment. Compared to the dust on Earth, Martian dust is much finer-grained, with the finest type only a few micrometers in diameter.⁶ Because of the fine grain size, the atmosphere contains some percentage of dust that is always suspended. This suspended dust affects the solar intensity and spectrum on Mars, which is important in the design of the solar cells which power many Mars missions including Pathfinder and the Mars Exploration Rovers (MER).⁵ Dust deposition has a large effect on mission lifetime, since dust building up on the solar cells decreases the power output of the solar cells, which is a limiting factor on the mission lifetime. The Materials Adherence Experiment (MAE) on Pathfinder found that dust accumulates on surfaces on Mars at a rate of about 0.28% per day, causing an estimated 0.29% decrease per day in power output by the solar arrays.⁴

In particular, one salient feature of the dusty Mars environment is the dust devils which appear in seasonal concentration, most frequently during the Martian spring and summer. Dust devils are swirling vortices of air that lift up and carry dust particles. Dust devils are an interesting phenomenon from the scientific perspective of understanding the Mars environment, as well as for the beneficial effect of extending mission lifetime when dust devils lift dust off of the solar arrays. The form of the dust devils varies widely in size, speed, time till dissipation, and dust concentration. *Greeley et al. [2006]* contains data for these dust devil parameters and more, while this paper focuses on analyzing the dust concentration distribution of the dust devils photographed by Mars Exploration Rover Spirit's Navcam.

* 2007 NASA Academy at Glenn Research Center, Space Photovoltaics, 302-1.

II. Problem Definition

Understanding the properties of dust devils is important for gaining knowledge of the Martian environment. The goal of this project is to investigate the distribution of dust concentration in dust devils on Mars, using the Navcam images taken by Spirit. The dust concentration of the dust devils is examined by determining the opacity, τ , which is a non-dimensionalized parameter describing the amount of dissipation that the dust devil causes to light traveling through it.

The distribution of τ was investigated to determine how τ varies with position within the dust devils, and how τ changes with time as the dust devils develop. One challenge was distinguishing the dust devils in the images from the background for the portion of the dust devil in front of the sky. Also of interest is how the optical properties of the dust in the devil compare to those of the dust suspended in the atmosphere; calculations were done first assuming the same optical properties and then more rigorously without these assumptions.

III. Approach

The Navcam images used to analyze the Mars dust devils are 21-frame sets of images which are taken over a time period ranging from 7 to 20 minutes. Each pixel of the image is a measurement of the line-of-sight light intensity from that area to the camera. Since τ is a measure of the dissipation of light by the dust devil, it can be calculated as follows, essentially comparing the intensity of light traveling through the dust devil with the intensity of light not traveling through the dust devil.

A. Optical Analysis Background and Calculating τ

A model of the dissipation resulting from traveling through a given medium is given by Beer's Law:

$$I = I_0 e^{-\tau}, \quad (1)$$

where I is the observed light intensity and I_0 is the initial light intensity. *Greeley et al* discusses the optical analysis methods used to determine τ of the dust devil. The calculations are essentially a matter of comparing the intensity of the light which travels through the dust devil with the intensity of the light which does not travel through the dust devil. This is done by looking at two lines of sight, one through the dust devil to the ground, the other directly to the ground without looking through the dust devil. The difference in light intensity for the line of sight through the dust devil is the result of light reflected from the ground being scattered out of the line of sight by the dust devil, as well as light being scattered in from the sky. Assuming a weighted combination of light reflected or scattered from the ground and the sky, the intensity of the dust devil line of sight is described as:

$$I_d = I_s + (I_g - I_s)e^{-\tau}, \quad (2)$$

where I_d is the intensity of the line of sight through the dust devil to the ground, I_s is the light intensity of the sky, and I_g is the light intensity of the line of sight to the ground not through the dust devil.² From Eq. (2), τ is obtained by the following rearrangement:

$$\tau = \ln[(I_g - I_s) / (I_d - I_s)], \quad (3)$$

Figure 1 shows where each of these intensity samples is taken from a typical dust devil image. The additional sample I_{sd} is the light intensity of the line of sight looking through the dust devil to the sky, instead of the ground.

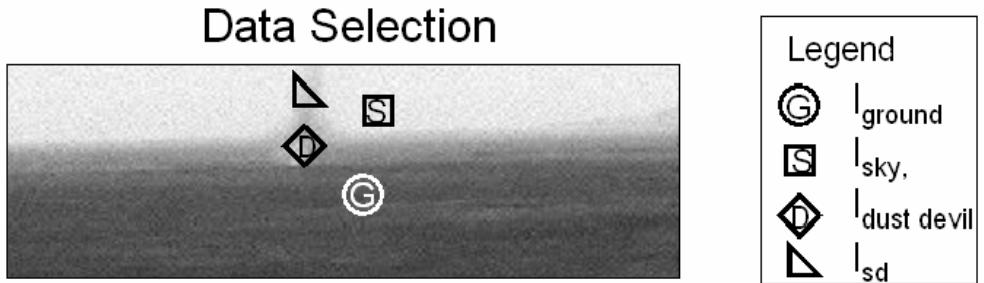


Figure 1. Intensity Sample Selection.

Equations (2) and (3) use the simplifying assumption of the same scattering properties for the dust in the dust devils and the dust in the atmosphere. To calculate τ without this assumption, an additional iteration of calculating τ is performed, using the light intensity sample I_{sd} to calculate I_{dd} , the intensity of an optically thick dust devil, meaning when $\tau \gg 1$:

$$I_{dd} = (I_{sd} - I_s e^{-\tau}) / (1 - e^{-\tau}). \quad (4)$$

This value of I_{dd} is then used in place of I_s , in Eq. 3, to calculate the new τ .²

B. Data Selection

As shown in Fig. 1, intensity samples of varying types need to be chosen from the dust devil images in order to calculate τ of the dust devil. For a given dust devil image, a sample of I_d is taken as a pixel in the dust devil. The corresponding value of I_g is taken from the same pixel in a previous frame so as to get a sample of the ground intensity without the dust devil. An average of an area in the sky is taken for I_s .

Selecting which pixels of the image to use for the intensity values is not a trivial matter, since the dust devil does not have clear boundaries and the sky and ground blur together at the horizon. Additionally, without any image processing or enhancement, it is usually very difficult to discern the dust devil directly from the radiometrically-calibrated images, as shown in Fig. 2.

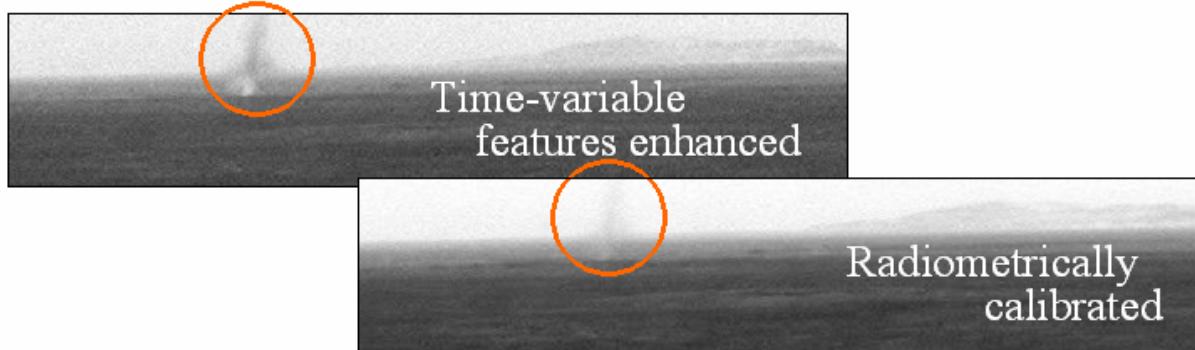


Figure 2. Comparison of Radiometrically-Calibrated and Enhanced Image.

The process of selecting pixels by looking at the enhanced image, and then getting the actual data from the radical image, is a tedious task that is easily automated. A simple java program and graphical user interface (GUI) provide graphic control of tasks including selecting areas of pixels, displaying the intensity value of a pixel, saving data in an excel-readable format, saving averages for blocks of pixels, indicating whether saved data is I_g , I_s , or I_d , and allowing comments to be saved. A screenshot of the GUI is shown in Fig. 3, and the code is appended in Appendix A.

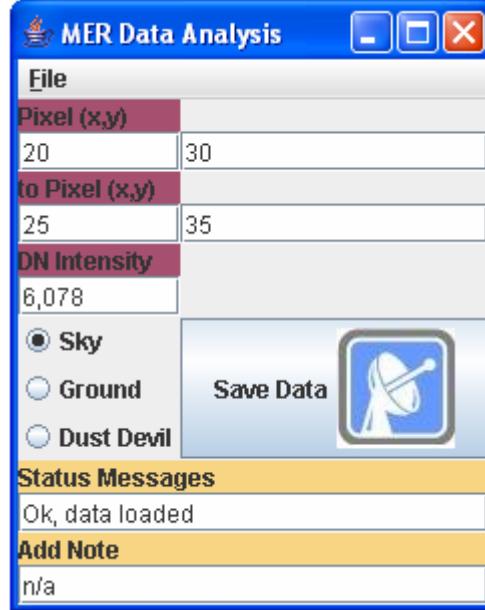


Figure 3. GUI for selecting and saving data from the images.

C. Initial Results

Initial calculations of τ were done in excel using the data collected using the developed GUI. The orange line in Fig. 4a shows a horizontal line sample of dust devil pixels which were sampled in order to examine the horizontal distribution of τ in the dust devil. Figure 4b is a graph of the τ values versus the horizontal x-position of the pixel along the sample of the dust devil. The edges of the sample show a noisy distribution of τ centered around 0.25, while the center of the dust devil varies only slightly around 0.19. These results are reasonable considering as a reference the values of τ previously obtained by Greeley et al in the range of 0.01 to 2.64 with a mean value of 0.14.²

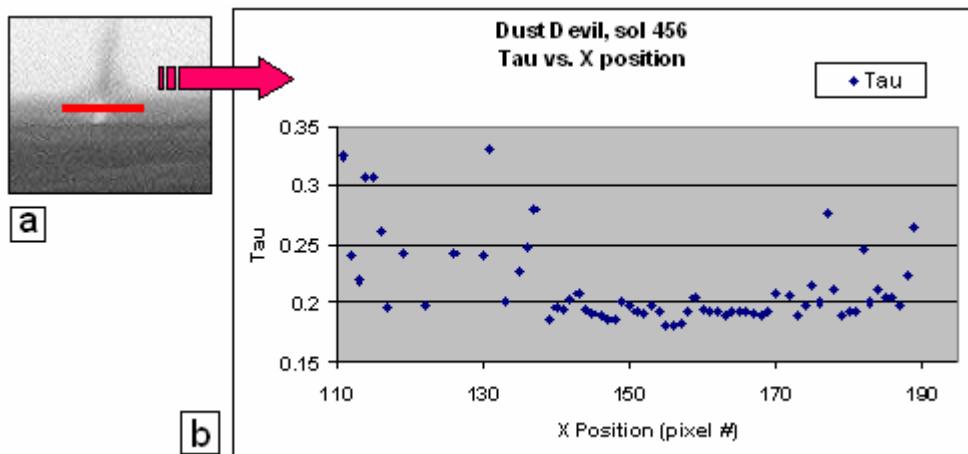


Figure 4. Typical Horizontal Distribution of Dust Devil τ . (a) illustration of sample selection parallel to the ground, (b) initial results showing horizontal distribution of τ in a typical dust devil.

These initial results, however, do not reveal much interesting information about the distribution of τ within the dust devil structure, so the next step was to develop a way of doing calculations and organizing the data in a more intuitive fashion.

D. Visualization of the Data

The first step in creating informative visualizations of the data was to be able to make new images of the dust devils. While excel had previously been used to do calculations and create graphs, the calculations were again coded in java so that entire images could be processed and a value of τ calculated for each pixel of the frame. Then some frame subtraction filtering would indicate which pixels had changed from the previous frame and were thus part of the dust devil. Then false color would be used to indicate the value of τ for each pixel of the dust devil in an image, so that the distribution of τ is visible by simply looking at the color contours. Appendix B contains the java code used to encode images in the .PNG format.

E. Making New Images

Figure 5 shows three frames of images from sol 456, demonstrating the application of false color to highlight the distribution of tau in the dust devil. The color scale is dark blue for the lower values of tau, then light blue, green, yellow, and finally red for the highest values of tau. From Fig. 1a to Fig. 1c shows how the dust devil concentration changes as the dust devil develops. The red and yellow portions of the dust devil indicate the highest opacity, and thus concentration, of the dust, showing that the dust devil is most concentrated in the lower center and more concentrated towards the center than the outer edges.

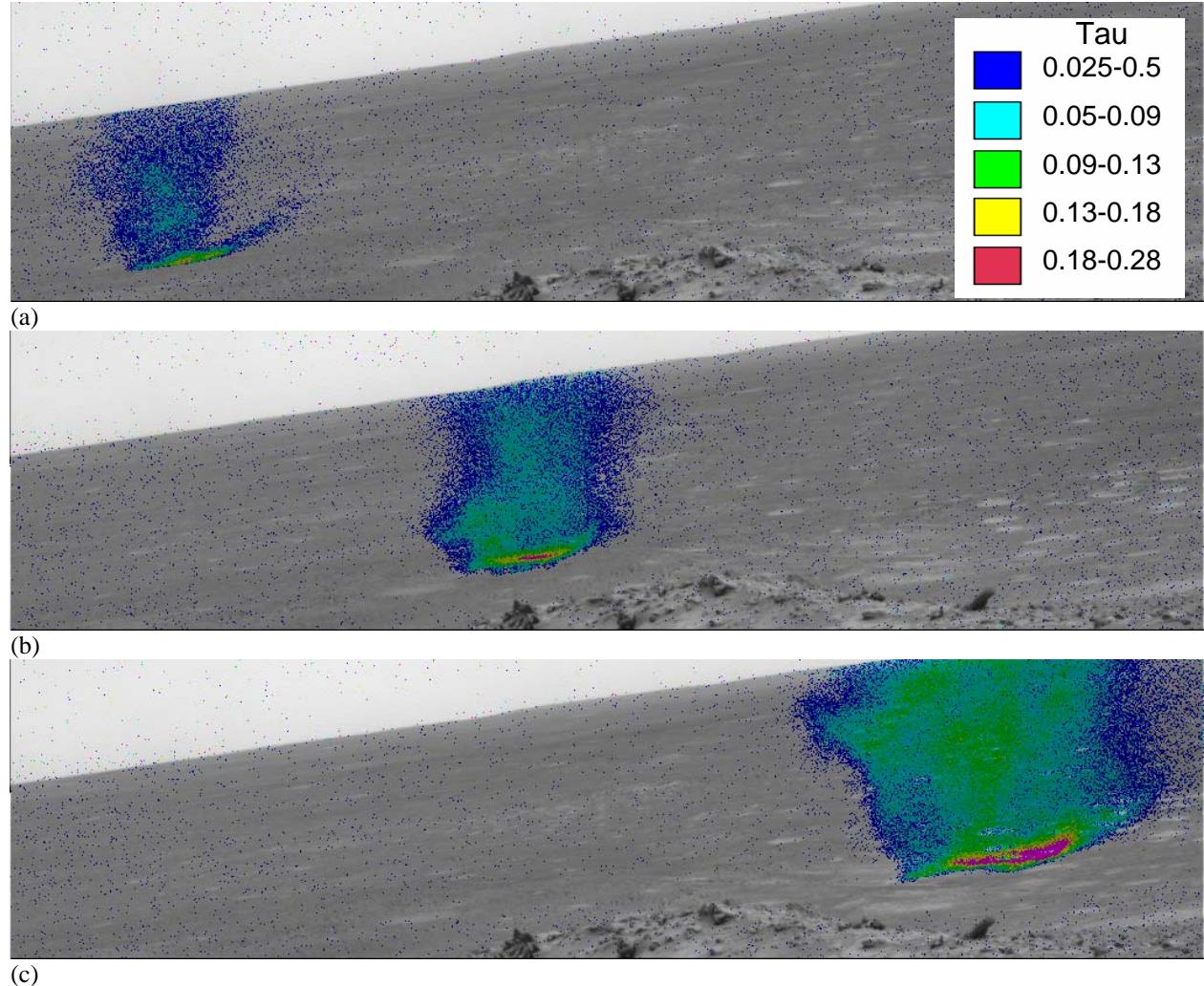


Figure 5. False-color images showing the distribution of τ in a dust devil from sol 456, frames 13-15.

To make these images, an earlier version of the program included in Appendix C took as input the original radiometrically-calibrated Spirit MER Navcam images. A first pass of processing the image containing the dust devil checked each pixel of the image to see if the intensity value was over 5000, and if so used the value to

calculate an average sky intensity to be used as I_s . The second pass of processing considered each pixel in the dust devil frame as I_g , and the same pixel in the previous frame as $I_{g'}$. These intensity values were used to calculate τ for each pixel.

After a τ value was calculated, that pixel was written in the new visualization image using an algorithm to display the dust devil in false color and the background in the normal grayscale. This was done by only displaying in color the pixels which had intensity values significantly different from the previous frame, which means the pixel had moved and was probably the dust devil. Since the calculations for τ were done for each pixel in the image, regardless of whether the pixel was in the sky or would be a bad sample because it was not clearly sky or ground, some τ values were undefined or less than zero and consequently discarded. Pains were taken to remove noise from the background, which was pixels that appeared to change between the previous frame and the dust devil frame, but only due to imprecise measurements. The noisy pixels were distributed at random in the image and did not appear to be part of the dust devil. They were almost completely removed by requiring a larger minimum difference between the previous frame and the dust devil frame in order to display a pixel in color as part of the dust devil.

After these initial results were obtained, the more rigorous calculations not assuming the same optical properties for the dust in the dust devil and the dust suspended in the sky, including Eqn. (4), were implemented. This allowed the discernment of the dust devils from the sky in the visualization images, which was not possible in the previous implementation, and is subtle but noticeable in Fig. 5b, where the dust devil appears to just end at the horizon.

In the new method of calculating τ , a first processing pass was used to get both an average I_{sd} as well as I_s . The average I_{sd} was obtained from criteria for pixels of an intensity greater than 4750, to indicate sky, and a minimum difference from the previous frame, to indicate dust devil. A new second pass was used to calculate an initial τ for each pixel, and an average τ for the dust devil to be used as a baseline value. Then the calculations using Egn. 4 were carried out to obtain the new values of τ .

IV. Results

Results from using the second, more rigorous, calculations of τ are shown in Fig. 6, which is a series of the first three frames of sol 640. The dust devil in this sequence is largely in front of the sky which gives a good opportunity for examining the difference between the optical properties of the dust devil and the sky. The images in Fig. 6 display the τ values of the sky as well as the dust devil, for comparison.

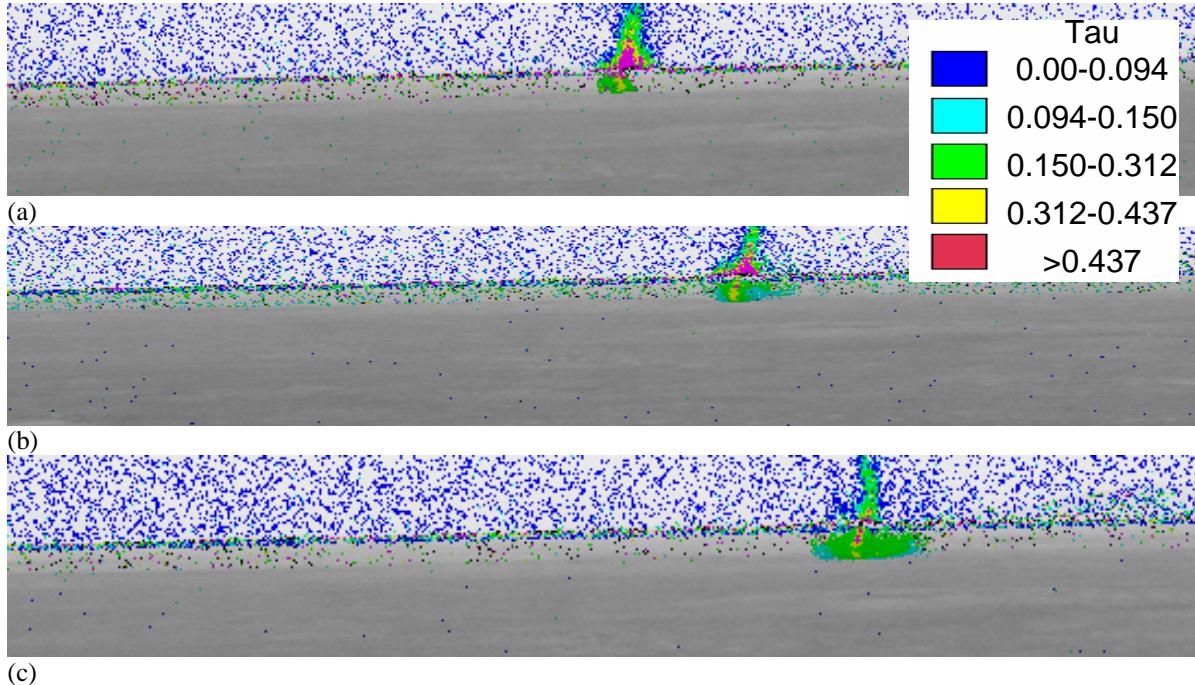


Figure 6. False-color images showing the distribution of τ in a dust devil from sol 640, frames 1-3, with the τ values of the sky displayed as well.

Table 1 shows the average dust devil and sky τ values calculated for the first four frames of sol 640. The last column of Table 1 is the difference between the dust devil and sky τ averages. Since looking at the images shows the dust devil as higher opacity than the sky, one would expect the average dust devil τ value to be slightly but consistently higher than the average sky τ value. An earlier round of results showed some average values of sky τ higher than average dust devil τ , but this was most likely due to a mistake in the average calculation. The mistake was probably using a cutoff intensity value to distinguish the sky which was not high enough, and so it allowed near-horizon pixels which may have been far-away ground, to be included in the sky average. Changing the cutoff values used to distinguish the sky from the rest of the frame resulted in the values in Table 2. Table 2 values show dust devil τ averages for the first four frames of sol 640 closer to the type of values expected, slightly higher than the τ averages of the sky in the frames.

Table 1. Average dust devil and sky tau values and difference between the two.

sol	frame	Avg Dust Devil Tau	Avg Sky Tau	Tau Difference
640	1	0.12263692	0.1281357	-0.0055
640	2	0.13820017	0.2116821	-0.07348
640	3	0.19402165	0.177779	0.016243
640	4	0.13292625	0.0897636	0.043163
Average:		0.14285964	0.1494181	-0.00656

Table 2. Average dust devil and sky tau values and difference between the two.

Using different filtering cutoff values to select sky pixels.

sol	frame	Avg Dust Devil Tau	Avg Sky Tau	Tau Difference
640	1	0.12123305	0.0409233	0.08031
640	2	0.13287802	0.0666786	0.066199
640	3	0.18782574	0.0828736	0.104952
640	4	0.12940697	0.0303151	0.099092
Average:		0.14283594	0.0551977	0.087638

V. Conclusion

In conclusion, the method of processing entire Navcam image frames was useful for looking at more data and getting averages of more samples than without using the developed software tools. However, filtering the frames to distinguish sky, ground, dust devil in front of ground, and dust devil in front of sky is not a highly robust process and could use additional refinement. This is evident in the variance of average dust devil and sky τ values from Table 1 to Table 1 as a result of changing cutoff values that select pixel samples for the sky and dust devil. Dust devil τ values were more stable than sky τ values. For the different cutoff filter values of sky intensity, the average dust devil average τ was almost the same, while average sky τ values varied more from an average of 0.149 to 0.0552 which is a large difference.

Acknowledgments

R. A. Author thanks mentor Geoffrey Landis for his help and support of the project. Also Dr. M. David Kankam, Director of the NASA Academy at Glenn, Michael Lamberty and Kamara Brown, the NASA Glenn Academy staffers for their hard work and support throughout the program. Thanks to the NASA Academy program for the opportunities provided, and to the Massachusetts Space Grant for sponsorship.

References

- ¹ Arvidson, R.E. et al., ‘Overview of the Spirit Mars Exploration Rover Mission to Gusev Crater: Landing site to Backstay Rock in the Columbia Hills.,’ *Journal of Geophysical Research*. Vol. 111. 2006.
- ² Greeley, Ronald et al., ‘Active Dust Devils in Gusev Crater, Mars.’ *Journal of Geophysical Research*. Vol. 111. 2006.
- ³ Landis, G., ‘Mars Dust-Removal Technology.’ *Journal of Propulsion and Power*. Vol. 14. 1998.

⁴ Landis, G., ‘Measurement of the Settling Rate of Atmospheric Dust on Mars by the MAE instrument on Mars Pathfinder. Journal of Geophysical Research, Vol. 111. 2000.

⁵ Landis, G., ‘Mars Solar Power.’ NASA/TM-2004-213367, 2004.

⁶ Lemmon, M.T. et al., “Atmospheric imaging results from the Mars Exploration Rovers: Spirit and Opportunity,” *Science*, Issue 306, pp. 1753-1756.

Appendix A. Java GUI code.

F. Main method

```
import java.io.*;
import java.io.IOException;

public class GUI {

    public static void main(String args[]) throws IOException
    {
        javax.swing.SwingUtilities.invokeLater(new Runnable()
        {
            public void run() {
                Display.DisplayGUI();
            }
        });
        Display.dp.rw.readFile(new File("2N166754107MRDA9C2P1560L0M1.img"));
        DisplayPane.tf6.setValue("Ok, data loaded");
    }
}
```

G. Display class

```
import java.awt.*;
import java.awt.event.*;
import java.io.FileWriter;
import java.io.IOException;
import javax.swing.*;
import java.text.*;

public class Display {
    static JFrame frame;
    static DisplayPane dp = new DisplayPane();

    public static void DisplayGUI() {
        frame = new JFrame("MER Data Analysis");
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.addWindowListener(new WindowAdapter() {
            public void windowClosing(WindowEvent ev) {
                try {
                    DisplayPane.writer.close();
                } catch (IOException e) {
                    e.printStackTrace();
                }
                System.exit(0);
            }
        });
        // Add to JPanel ContentPane
        //addComponentsToPane(frame.getContentPane());
        frame.add(dp);
    }
}
```

```

// Add menu bar
frame.setJMenuBar(DisplayPane.menuBar);

frame.pack();
//int width=500;
//int height=500;
//frame.setSize(width, height);
frame.setVisible(true);
}
}

```

H. DisplayPane class

```

import java.io.*;
import java.awt.Color;
import java.awt.ComponentOrientation;
import java.awt.Dimension;
import java.awt.GridBagConstraints;
import java.awt.GridBagLayout;
import java.awt.event.KeyEvent;
import java.beans.PropertyChangeListener;
import java.beans.PropertyChangeEvent;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.awt.event.ItemEvent;
import java.awt.event.ItemListener;
import javax.swing.*;
//import javax.swing.filechooser.*;

public class DisplayPane extends JPanel
    implements PropertyChangeListener, ActionListener,
    ItemListener {

    public RW rw;
    final static boolean shouldWeightX = true;
    final static boolean RIGHT_TO_LEFT = false;
    static boolean linear = false;
    final JFileChooser fc;
    static JMenuBar menuBar;
    static JCheckBoxMenuItem menuCheckBox;
    public static JTextField tf1, tf2, tf3, tf4, tf5, tf6, tf7;
    File file=new File("2N166754107MRDA9C2P1560L0M1.img"); // default file
    //File multFiles[];
    //int numImages = 21;
    static int pixelXValue = 1;
    static int pixelYValue = 1;
    static int toX = 1;
    static int toY = 1;
    static int dnValue = 1;
    static FileWriter writer;
    static String pixelType = "S"; // default
    static String note = " ";

    public DisplayPane() {
        rw = new RW();
        if (RIGHT_TO_LEFT) {

```

```

        setComponentOrientation(ComponentOrientation.RIGHT_TO_LEFT );
    }

        setLayout(new GridBagLayout());
GridBagConstraints c = new GridBagConstraints();

c.fill = GridBagConstraints.HORIZONTAL;
if (shouldWeightX) {
    c.weightx = 0.5;
}

//open writer
try {
    writer = new FileWriter("dataProc.txt");
    writer.write("Image File"+';'+ "Sky/Ground/Dust"+';
                +"X"+';'+ "Y"+';'+ "I"+';'+ "Notes"+'\n');
} catch (IOException e) {
    e.printStackTrace();
}

// Messy menu stuff
menuBar = new JMenuBar();
menuBar.setOpaque(true);
//menuBar.setBackground(new Color(154, 120, 127));
menuBar.setPreferredSize(new Dimension(200, 20));
JMenu menu = new JMenu("File");
menu.setMnemonic(KeyEvent.VK_F);
menuBar.add(menu);
JMenuItem menuOpen = new JMenuItem("Open", new ImageIcon("icon2.jpg"));
menuCheckBox = new JCheckBoxMenuItem("linear data set b/w selected pixels");
menuCheckBox.getAccessibleContext().setAccessibleDescription(
    "Get a line of data points between the two specified " +
    "pixel locations, instead of a block of data points");
menu.add(menuOpen);
menu.add(menuCheckBox);

menuOpen.setActionCommand("open_file");
menuOpen.addActionListener(this);
menuCheckBox.addItemListener(this);
fc = new JFileChooser();
//fc.setFileSelectionMode(JFileChooser.FILES_AND_DIRECTORIES);

// Labels
JLabel label1 = new JLabel("Pixel (x,y)");
JLabel label2 = new JLabel("to Pixel (x,y)");
JLabel label3 = new JLabel("DN Intensity");
JLabel label4 = new JLabel("Status Messages");
JLabel label5 = new JLabel("Add Note");
label1.setOpaque(true);
label2.setOpaque(true);
label3.setOpaque(true);
label4.setOpaque(true);
label5.setOpaque(true);
label1.setBackground(new Color(164, 80, 110));
label2.setBackground(new Color(164, 80, 110));
label3.setBackground(new Color(164, 80, 110));

```

```

label4.setBackground(new Color(248, 213, 131));
label5.setBackground(new Color(248, 213, 131));

// Text fields
tf1 = new JFormattedTextField();
tf2 = new JFormattedTextField();
tf3 = new JFormattedTextField();
tf4 = new JFormattedTextField();
tf5 = new JFormattedTextField();
tf6 = new JFormattedTextField();
tf7 = new JFormattedTextField();
tf1.setValue(pixelXValue);
tf2.setValue(pixelYValue);
tf3.setValue(toX);
tf4.setValue(toY);
tf5.setValue(dnValue);
tf6.setValue("Smooth sailing...\"");
tf7.setValue("n/a");
tf1.addPropertyChangeListener("value",this);
tf2.addPropertyChangeListener("value",this);
tf3.addPropertyChangeListener("value",this);
tf4.addPropertyChangeListener("value",this);
tf7.addPropertyChangeListener("value",this);

// Button
ImageIcon GoButtonIcon = new ImageIcon("space_icon.jpg");
JButton b1 = new JButton("Save Data", GoButtonIcon);
b1.setVerticalTextPosition(AbstractButton.CENTER);
b1.setHorizontalTextPosition(AbstractButton.LEADING);
b1.setMnemonic(KeyEvent.VK_G);
b1.setToolTipText("Do some data analysis...\"");
b1.setActionCommand("save");
b1.addActionListener(this);

// Radio buttons
JRadioButton r1 = new JRadioButton("Sky");
r1.setSelected(true);
JRadioButton r2 = new JRadioButton("Ground");
JRadioButton r3 = new JRadioButton("Dust Devil");
ButtonGroup group = new ButtonGroup();
group.add(r1);
group.add(r2);
group.add(r3);
r1.addActionListener(this);
r2.addActionListener(this);
r3.addActionListener(this);
r1.setActionCommand("S");
r2.setActionCommand("G");
r3.setActionCommand("D");

c.gridx = 0;
c.gridy = 0;
add(label1, c);

c.gridy = 1;

```

```

        add(tf1, c);

        c.gridx = 1;
        add(tf2, c);

        c.gridx = 0;
        c.gridy = 2;
        add(label2, c);

        c.gridx = 0;
        c.gridy = 3;
        add(tf3, c);

        c.gridx = 1;
        add(tf4, c);

        c.gridx = 0;
        c.gridy = 4;
        add(label3, c);

        c.gridy = 5;
        add(tf5, c);

        c.gridy = 6;
        add(r1,c);

        c.gridy = 7;
        add(r2,c);

        c.gridy = 8;
        add(r3,c);

        c.fill = GridBagConstraints.NONE;
        c.gridheight = 3;
        c.gridx = 1;
        c.gridy = 6;
        add(b1, c);

        c.fill = GridBagConstraints.HORIZONTAL;
        c.gridheight = 1;
        c.gridwidth = 2;
        c.gridx = 0;
        c.gridy = 9;
        add(label4, c);

        c.gridy = 10;
        add(tf6, c);

        c.gridy = 11;
        add(label5,c);

        c.gridy = 12;
        add(tf7, c);
    }

    public void propertyChange(PropertyChangeEvent e) {

```

```

Object source = e.getSource();

if (source == tf1) {
    pixelXValue = ((Number)tf1.getValue()).intValue();
    tf3.setValue(pixelXValue);
    if (pixelXValue>0 && pixelXValue<1024)
        tf5.setValue(rw.image[pixelXValue-1][pixelYValue-1]);
}
else if (source == tf2) {
    pixelYValue = ((Number)tf2.getValue()).intValue();
    tf4.setValue(pixelYValue);
    if (pixelYValue>0 && pixelYValue<256)
        tf5.setValue(rw.image[pixelXValue-1][pixelYValue-1]);
}
else if (source == tf3) {
    toX = ((Number)tf3.getValue()).intValue();
}
else if (source == tf4) {
    toY = ((Number)tf4.getValue()).intValue();
}
else if (source == tf7) {
    note = (String)tf7.getValue();
}
}

public void actionPerformed(ActionEvent e){
    // Save button
    if ("save".equals(e.getActionCommand())) {
        writeDataSet();
    }
    // Menu-open
    else if ("open_file".equals(e.getActionCommand())) {
        int returnVal = fc.showOpenDialog(DisplayPane.this);

        if (returnVal == JFileChooser.APPROVE_OPTION) {
            file = fc.getSelectedFile();
            if(file.isFile()){
                try {
                    rw.readFile(file);
                } catch (IOException e1) {
                    // TODO Auto-generated catch block
                    e1.printStackTrace();
                }
            }
            tf6.setValue("loaded"+file);
        }
    }
    // Radio button to select dust devil, ground, or sky identifier
    else pixelType = e.getActionCommand();
}

public void itemStateChanged(ItemEvent e) {
    if (e.getItemSelectable() == menuCheckBox)
        linear = !linear;
}

```

```

///////////
// writeDataSet writes (a) line(s) of the following data:
// image file, dust/sky/ground identifier, pixel X, pixel Y,
// intensity value, and user note
// NOTE: image array indices range [0-1023][0-255]
// to access pixel x,y want image[x-1][y-1]
///////////

public void writeDataSet() {

    if (toX>1024 || toY>256) {
        tf6.setValue("requested x/y out of bounds");
        return;
    }
    int average = 0;
    int imageValue;
    try {
        if (pixelXValue == toX && pixelYValue==toY)
            writer.write(file.getName()+';'+pixelType+';'+(pixelXValue)
                        +'_'+(pixelYValue)+';'+tf5.getValue()+';'+note+'\n');
        else if (linear == false){
            for (int y=pixelYValue; y <= toY; y++)
            {
                for (int x=pixelXValue; x <= toX; x++)
                {
                    //tf5.setValue(RW.image[x-1][y-1]);
                    imageValue = rw.image[x-1][y-1];
                    writer.write(file.getName()+';'+pixelType+';'+(x)
                                +'_'+(y)+';'+imageValue+'\n');
                    average += imageValue;
                }
            }
            average /=( (toX-pixelXValue+1)*(toY-pixelYValue+1) );
            writer.write(file.getName()+';'+pixelType+';'+(pixelXValue+"_"+toX)
                        +'_'+(pixelYValue+"_"+toY)+';'+average+';'+average intensity+';'+note+'\n');
        }
        else {
            float m = (float)(toY-pixelYValue)/(float)(toX-pixelXValue);
            //float b = -(float)toY/(float)(m*toX);
            //System.out.println("m "+m);
            float y = pixelYValue;
            for (int x=pixelXValue; x <= toX; x++)
            {
                //y = (int)(m*(float)x + b);
                y = (int)(pixelYValue+m*(float)(x-pixelXValue));
                //System.out.println("x "+x+" y "+y);
                imageValue = rw.image[x-1][(int)y-1];
                writer.write(file.getName()+';'+pixelType+';'+(x)
                            +'_'+(y)+';'+imageValue+';'+note+'\n');
            }
        }
    } catch (IOException e) {
        e.printStackTrace();
    }
}
}

```

Appendix B. Java .PNG Image Encoder code.

```
import java.io.ByteArrayOutputStream;
import java.io.FileNotFoundException;
import java.io.FileOutputStream;
import java.io.IOException;
import java.util.zip.CRC32;
import java.util.zip.Deflater;

///////////////////////////////
/* Encode .PNG extension image files
 *
 * PNG format:
 * 1. PNG file signature
 * 2. IHDR (image header) chunk
 * 3. IDAT (compressed image data) chunk
 * 4. IEND
 * Each chunk: length (4 bytes), type name (4 bytes),
 * data (n bytes), CRC (4bytes)
 *////////////////////////////

public class PNG_Encoder {

    FileOutputStream writer;
    CRC32 CRC;
    // Table of CRCs of all 8-bit messages
    static int[] crc_table = new int[256];
    static boolean crc_table_computed = false;
    Deflater Compressor;
    // IHDR things
    public static final byte[] PNG_FILE_SIG = {(byte)0x89, (byte)0x50, (byte)0x4e, (byte)0x47,
                                                (byte)0x0d, (byte)0xa, (byte)0xa, (byte)0xa};
    public static final byte[] IHDR_LENGTH = {0,0,0,13};
    public static final int BIT_DEPTH = 0x08; // 8 bits per pixel
    public static final int COLOR_TYPE = 0x02; // 2 means RGB
    public static final int COMPRESSION_MODE = 0x0; // 0 only defined mode
    public static final int FILTER_METHOD= 0x0;
    public static final int INTERLACE_METHOD= 0x0;
    public static final byte[] IHDR_TYPE_DATA = {(byte)'I',(byte)'H',(byte)'D',(byte)'R',
                                                0x0,0x0,0x04,0x0, // image width
                                                0x0,0x0,0x01,0x0, // image height
                                                (byte)BIT_DEPTH, (byte)COLOR_TYPE,
                                                (byte)COMPRESSION_MODE, (byte)FILTER_METHOD,
                                                (byte)INTERLACE_METHOD};

    public static final byte[] IEND_LENGTH = {0,0,0,0};
    public static final byte[] IEND_TYPE = {(byte)'I',(byte)'E',(byte)'N',(byte)'D'};
    public static final byte[] IDAT_LENGTH = {0x0,(byte)0xC,0x1,0x4}; //786688
    public static final byte[] PHYS_LENGTH = {0x0, 0x0, 0x0, (byte)0x09};
    public static final byte[] PHYS_TYPE_DATA = {(byte)'p',(byte)'H',(byte)'Y',(byte)'s',
                                                0x0,0x0,(byte)0x1B,(byte)0xAF, //pixels per xunit
                                                0x0,0x0,(byte)0x1B,(byte)0xAF, //pixels per yunit
                                                (byte)0x01};

    public static final byte[] GAMA_LENGTH = {0x0,0x0,0x0,0x4};
    public static final byte[] GAMA_TYPE_DATA = {(byte)'g',(byte)'A',(byte)'M',(byte)'A',
                                                0x0,0x0,(byte)0xB1,(byte)0x8E};
    public static final byte[] CHRM_LENGTH = {0x0,0x0,0x0,(byte)0x20};
```

```

public static final byte[] CHRM_TYPE_DATA = {(byte)'c',(byte)'H',(byte)'R',(byte)'M',
                                             0x0,0x0,(byte)0x7A,(byte)0x25,
                                             0x0,0x0,(byte)0x80,(byte)0x83,
                                             0x0,0x0,(byte)0xF9,(byte)0xFF,
                                             0x0,0x0,(byte)0x80,(byte)0xE9,
                                             0x0,0x0,(byte)0x75,(byte)0x30,
                                             0x0,0x0,(byte)0xEA,(byte)0x60,
                                             0x0,0x0,(byte)0x3A,(byte)0x98,
                                             0x0,0x0,(byte)0x17,(byte)0x6F};

///////////////////////////////
// constructor:
// 1. open file to write as .PNG
// 2. write header info (file sig, ihdr)
// 3. make CRC lookup table
/////////////////////////////
public PNG_Encoder() {
    CRC = new CRC32();
    Compressor = new Deflater();
    try {
        writer = new FileOutputStream("processed_image.png");
    } catch (FileNotFoundException e) {
        e.printStackTrace();
    }
    try {
        writer.write(PNG_FILE_SIG);
        writer.write(IHDR_LENGTH);
        writer.write(IHDR_TYPE_DATA);
        CRC(IHDR_TYPE_DATA);
        writer.write(PHYS_LENGTH);
        writer.write(PHYS_TYPE_DATA);
        CRC(PHYS_TYPE_DATA);
        writer.write(GAMA_LENGTH);
        writer.write(GAMA_TYPE_DATA);
        CRC(GAMA_TYPE_DATA);
        writer.write(CHRM_LENGTH);
        writer.write(CHRM_TYPE_DATA);
        CRC(CHRM_TYPE_DATA);
    } catch (IOException e) {
        e.printStackTrace();
    }
}

/////////////////////////////
// write image data and CRC values
// For 8-bit samples (3 bytes per pixel) should be total of 786,436 bytes
// (including 4 bytes for IDAT header)
// Inputs: byte array [786426] with 4-byte IDAT type, plus RGB image data
// Outputs: none
/////////////////////////////
public void writePNG(byte[] imageData) {
    try {
        Compressor.setInput(imageData);
        Compressor.finish();
    }

    // expandable byte array to hold the compressed data.
}

```

```

// can't use an array that's the same size as the orginal because
// there is no guarantee that the compressed data will be smaller than
// the uncompressed data.
ByteArrayOutputStream bos = new ByteArrayOutputStream(imageData.length);

// Compress the data
byte[] buf = new byte[1024];
while (!Compressor.finished()) {
    int count = Compressor.deflate(buf);
    bos.write(buf, 0, count);
}
try {
    bos.close();
} catch (IOException e) {
}
byte[] compressedData = new byte[bos.size()+4];
compressedData[0]=(byte)'T';
compressedData[1]=(byte)'D';
compressedData[2]=(byte)'A';
compressedData[3]=(byte)'T';
System.arraycopy(bos.toByteArray(),0,compressedData,4,bos.size());

// Write size of (compressed) IDAT field
for (int i=3;i>-1;i--) {
    writer.write( (bos.size()>>>i*8) & 0xFF);
}

// Write IDAT
writer.write(compressedData);
CRC(compressedData);
} catch (IOException e) {
}
}

///////////////////////////////
// closePNG writes IEND chunk and closes writer
/////////////////////////////
public void closePNG() {
    try {
        writer.write(IEND_LENGTH);
        writer.write(IEND_TYPE);
        CRC(IEND_TYPE);
        writer.close();
    } catch (IOException e) {
        e.printStackTrace();
    }
}

/////////////////////////////
// CRC writes the 4-byte CRC code
// Input: byte array of type followed by data
/////////////////////////////
public void CRC(byte[] data) {
    CRC.update(data);
    long temp= CRC.getValue();
    int crc = (int)(temp & 0xFFFFFFFF);
}

```

```

        for (int i=3; i>-1; i--) {
            byte tempByte = (byte)((crc>>(i*8)) & 0xFF);
            try {
                writer.write(tempByte);
            } catch (IOException e) {
                e.printStackTrace();
            }
        }
        CRC.reset();
    }
}

```

Appendix C. Java visualization image creation code.

```

import java.io.File;
import java.io.IOException;
import java.util.ArrayList;

public class TauDisplay {
    //static String backgroundFile1 = "456\\2N166841449MRDA9DWP1560L0M1.img";
    //static String backgroundFile2 = "456\\2N166841469MRDA9DWP1560L0M1.img";
    //static String dustDevilFile = "456\\2N166841529MRDA9DWP1560L0M1.img";
    static String backgroundFile1 = "640\\2N183183433MRDAI00P1560L0M1.img";
    static String backgroundFile2 = "640\\2N183183615MRDAI00P1560L0M1.img";
    static String dustDevilFile = "640\\2N183183311MRDAI00P1560L0M1.img";
    final static int imageArraySize = 786688;
    static byte[] imageData = new byte[imageArraySize];
    static RW rw1, rw2, rw3;
    static int Isky=0, Isd=0;
    static double cutoff1=0.1, cutoff2=0.22, cutoff3=0.35, cutoff4=0.5;

    /////////////////////////////////
    // Main Method
    ///////////////////////////////
    public static void main(String[] args) {
        PNG_Encoder PNG = new PNG_Encoder();
        rw1 = new RW();
        rw2 = new RW();
        rw3 = new RW();
        try {
            rw1.readFile(new File("C:\\Documents and Settings\\Beccs\\Desktop\\spirit pictures\\"+backgroundFile1));
            rw2.readFile(new File("C:\\Documents and Settings\\Beccs\\Desktop\\spirit pictures\\"+backgroundFile2));
            rw3.readFile(new File("C:\\Documents and Settings\\Beccs\\Desktop\\spirit pictures\\"+dustDevilFile));
        } catch (IOException e) {
            e.printStackTrace();
        }
        // Get Isky and Isd from image
        skyCalc(rw1.image, rw2.image, rw3.image);
        Tau(rw1.image, rw2.image, rw3.image);
        //subtractFrames(rw2.image, rw3.image);
        PNG.writePNG(imageData);
        PNG.closePNG();
    }

    ///////////////////////////////
    // Subtracts 2 frames and saves the resulting
    // image in imageData- the difference above a min. cutoff
    // is in color, the rest is grayscale
    ///////////////////////////////
}

```

```

public static void subtractFrames(int[][] image1, int[][] image2) {
    int i = 0; int a=0; int b=0;
    for (int j=0; j<256; j++) {
        imageData[i]=0x00; i++;
        for (int k=0; k<1024; k++) {
            a = image1[k][j];
            b = image2[k][j];
            int c = Math.abs(a-b);
            if (c<100) {
                imageData[i]=imageData[i+1]=imageData[i+2]=
                    (byte)((float)a/6500.0*255.0);
            }
            else if (c<130){
                imageData[i]=0x0;
                imageData[i+1]=(byte)((float)b/6500.0*255.0);
                imageData[i+2]=0x0;
            }
            else if (c<160){
                imageData[i]=0x0;
                imageData[i+1]=(byte)((float)b/6500.0*255.0);
                imageData[i+2]=(byte)((float)b/6500.0*255.0);
            }
            else if (c<200){
                imageData[i]=(byte)((float)b/6500.0*255.0);
                imageData[i+1]=0x0;
                imageData[i+2]=(byte)((float)b/6500.0*255.0);
            }
            else if (c>=200){
                imageData[i]=(byte)((float)b/6500.0*255.0);
                imageData[i+1]=0x0;
                imageData[i+2]=0x0;
            }
            i+=3;
        }
    }
}

///////////////////////////////
// skyCalc calculates Isd, intensity of sky through dust devil,
// and Isky, the intensity of sky light
//
// Debugging note: commented sections are for displaying
// image in teal, yellow, and pink to show which pixels
// are being counted as sky, dust devil in front of sky,
// and borderline sky not being used
/////////////////////////////
public static void skyCalc(int[][] image1, int[][] image2, int[][] image3) {
    int i=0, b=0, skyCount=0, skySampleCount=0, IdcCount=0;
    for (int j=0; j<256; j++) {
        imageData[i]=0x00; i++;
        skySampleCount=0;
        for (int k=0; k<1024; k++) {
            b = image3[k][j];
            if (b>4750) {
                skySampleCount++;
                // Sky without dust devil (teal)
                if (Math.abs(b-image2[k][j])<80 && b>5000) {
                    Isky+=b;
                    skyCount++;
                    //imageData[i]=0x0; imageData[i+1]=(byte)0xFF; imageData[i+2]=(byte)0xFF;
                }
                // Dust devil in front of sky (yellow)
            }
        }
    }
}

```

```

        else if (Math.abs(b-image2[k][j])>140) {
            Isd+=b;
            Iddcount++;
            //imageData[i]=(byte)0xFF; imageData[i+1]=(byte)0xFF; imageData[i+2]=0x0;
        }
        // borderline Sky not being counted (pink)
        /*else {
            imageData[i]=(byte)0xFF;
            imageData[i+1]=0x0;
            imageData[i+2]=(byte)0xFF;
        }*/
    }
    /*else {
        imageData[i]=0x0;
        imageData[i+1]=(byte)0xFF;
        imageData[i+2]=0x0;
    }*/
    // Stop if done with sky pixels
    if (skySampleCount < 3 && k>50) {
        Isky/=skycount;
        Isd=Iddcount;
        System.out.println("Isky "+Isky);
        System.out.println("Isd "+Isd);
        return;
    }
    i +=3;
}
//System.out.println("j "+j+" b "+b+" srow "+lastSkyRow);
}

////////////////////////////////////////////////////////////////
// Tau calculates dust devil tau values
// and saves an image to imageData
// color-coded for the tau's which correspond to the dust devil
// while the background is grayscale
////////////////////////////////////////////////////////////////
public static void Tau(int[][] image1, int[][] image2, int[][] image3) {
    int i=0, a=0, b=0, c=0, avgTauCount=0, SkyTauCount=0;
    double d=0.0, e=0.0, tau=0.0, Idd=0.0, avgDDTau=0.0, avgSkyTau=0.0;
    ArrayList<Double> groundTaus = new ArrayList();
    // Calculate tau for ground samples only (sky later)
    for (int j=0; j<256; j++) {
        for (int k=0; k<1024; k++) {
            if ((c = image3[k][j])<4750) {
                a = image1[k][j];
                b = image2[k][j];
                d = b-Isky;
                e = c-Isky;

                tau = (d!=0 && e!=0) ? Math.log(d/e) : 0.00;
                // Want to get avg dust devil tau for sky calculations later
                if ((Math.abs(b-c)>120) && (Math.abs(a-c)>120)) {
                    avgDDTau += tau;
                    avgTauCount++;
                }
                groundTaus.add(tau);
            }
        }
    }
    // Calculate cutoff values for display colors
}

```

```

// i.e. all tau values < cutoff1 will be one color
avgDDTau /= avgTauCount;
cutoff1 = 0.75*avgDDTau;
cutoff2 = 1.2*avgDDTau;
cutoff3 = 2.5*avgDDTau;
cutoff4 = 3.5*avgDDTau;
System.out.println("avg dust devil Tau "+avgDDTau);
System.out.println("color      cutoffs      (dark      blue,      light      blue,      green,      yellow,
red):"+'\n'+cutoff1+"," +cutoff2+"," +cutoff3+"," +cutoff4);

// Write image values in imageData array
int n=0;
double Idd_sky = (Isd-Isky*Math.exp(-avgDDTau))/(1-Math.exp(-avgDDTau));
for (int j=0; j<256; j++) {
    imageData[i]=0x00; i++;
    for (int k=0; k<1024; k++) {
        a = image1[k][j];
        b = image2[k][j];
        c = image3[k][j];
        // Use Idd instead of Isky- calc Idd for ground sample, use avg for sky sample
        if (c<4750) {
            Idd = (Isd-Isky*Math.exp(-groundTaus.get(n)))/(1-Math.exp(-groundTaus.get(n)));
            n++;
        }
        else
            Idd = Idd_sky;
        d = b-Idd;
        e = c-Idd;
        tau = (d!=0 && e!=0) ? Math.log(d/e) : 0.00;

        if (c>5000 && tau>0.00) {
            avgSkyTau+=tau;
            SkyTauCount++;
        }

        /* To qualify as dust devil, must have
         * 1) min. difference from two previous frames w/o dust devil
         * 2) tau value w/i (wide) range expected of dust devil
         */
        // Added "c<4000" to display grayscale for ground but not sky
        // if intensity >4000 probably sky so display tau
        if (((Math.abs(b-c)<120) && (Math.abs(a-c)<120) && c<4000)
            || tau<0.00001) {
            imageData[i]=imageData[i+1]=imageData[i+2]=(byte)((float)c/6300.0*255.0);
        }
        else if (tau<cutoff1){
            imageData[i]=0x0;
            imageData[i+1]=0x0;
            imageData[i+2]=(byte)((float)c/6500.0*255.0);
        }
        else if (tau<cutoff2){
            imageData[i]=0x0;
            imageData[i+1]=(byte)((float)c/6500.0*255.0);
            imageData[i+2]=(byte)((float)c/6500.0*255.0);
        }
        else if (tau<cutoff3){
            imageData[i]=0x0;
            imageData[i+1]=(byte)((float)c/6500.0*255.0);
            imageData[i+2]=0x0;
        }
        else if (tau<cutoff4){
            imageData[i]=(byte)((float)c/6500.0*255.0);
        }
    }
}

```

```
        imageData[i+1]=(byte)((float)c/6500.0*255.0);
        imageData[i+2]=0x0;
    }
    else if (tau>=cutoff4){
        imageData[i]=(byte)((float)c/6500.0*255.0);
        imageData[i+1]=0x0;
        imageData[i+2]=(byte)((float)c/6500.0*255.0);
    }
    i+=3;
}
avgSkyTau/=SkyTauCount;
System.out.println("avg sky tau "+avgSkyTau);
}
}
```